Optimal Acyclic Fine-Grain Scheduling with Cache Effects for Embedded and Real Time Systems

Sid-Ahmed-Ali Touati
INRIA. Domaine de Voluceau, BP 105
78153 Le Chesnay cedex, France
Sid-Ahmed-Ali.Touati@inria.fr

ABSTRACT

To sustain the increases in processor performance, embedded and real-time systems need to find the best total schedule time when compiling their application. The optimal acyclic scheduling problem is a classical challenge which has been formulated using integer programming in lot of works. In this paper, we give a new formulation of acyclic instruction scheduling problem under registers and resources constraints in multiple instructions issuing processors with cache effects. Given a direct acyclic graph $G = (V, E)$, the complexity of our integer linear programming model is bounded by $\mathcal{O}(|V|^2)$ variables and $\mathcal{O}(|E|+|V|^2)$ constraints. This complexity is better than the complexity of the existing techniques which includes a worst total schedule time factor.

Keywords

optimal acyclic schedule, registers constraints, resources constraints, cache effects, integer programming

1. INTRODUCTION

Current compilers try to take benefit from the instruction level parallelism (ILP) present in nowadays processors. Multiple operations are issued in the same clock cycle to increase the throughput of the executed operations. Completing a computation in the shortest time is a scheduling problem constrained by many factors. The most important ones are the data dependencies, the availability of the hardware features and the memory hierarchy constraints. This latter include the registers constraints and the cache effects. While the registers constraints impose the fact that the number of values simultaneously alive must not exceed the number of available registers, the cache effects are different : in fact, the caches misses are only a source of performance bottlenecks because a miss penalty may stall the processor. Furthermore, the cache behavior is difficult to predict statically, making the verification and optimization of real time application harder. In our formulation, we give a first approach to handle compulsory (cold start) cache misses where a the memory access operations exhibit some spatial or temporal locality [10].

The theoretical studies on scheduling reveal that integrating the resources constraints [2] or the registers constraints [4] are two NP-complete problems. Combining scheduling under both the registers and resources constraints become a complex task where the general compilers use some heuristics to get an optimized schedule in polynomial time complexity. However, the embedded and real time systems can require the optimal (best) schedule. We have to write a "good" formulation of the problem in order to reduce the resolution time. Many works have been done using integer linear programming (intLP) models [15, 7, 8, 9, 4, 1, 5, 3]. In our work, we present a new formulation of acyclic scheduling such that the complexity of the model generated is lower than these existing techniques while we include some cache optimizations, like we will explain in the end of this paper. Our formulation must reduce the resolution time since we considerably reduce the number of variables and constraints in the generated intLP model.

This paper is organized as following. We first present the model of the targeted processors in Sect. 2 and the direct acyclic graph (DAG) to be scheduled in Sect. 3 : in our study, we assume heterogeneous FUs, more than one register type, and delayed latencies of writing into and reading from registers. The problem of acyclic scheduling is briefly recalled in Sect. 4. After, we define some intLP modeling techniques in Sect. 5. We use these techniques to write our intLP formulation in Sect. 6. We present some achieved work in this field in Sect. 7 and conclude by our remarks and perspectives in Sect. 8.

2. PROCESSOR DESCRIPTION

An ILP processor [12] takes benefit from the inherent parallelism in the instructions flow and issues multiple operations per clock cycle thanks to the pipelined execution and the presence of multiple functional units (FUs). An operation can be executed on one (or more) functional units (FU). We model the complex behavior of the execution of the operations on the FUs by the reservation tables. We attach to each instruction a reservation table (RT) to describe at which clock cycle a FU is busy due to the execution of this instruction on it. A RT consists of a two-dimensional table, where the number of lines is the latency of the operation, and the columns consists of the set of FUs. Given a RT of an instruction u, $\mathcal{RT}_u(c, q) = 1$ means that u executes on

the FU q during the clock cycle c after its issuing.

The target processor $\mathcal{P}$ is described by $\mathcal{T}$ the set of its registers types (float, int, etc.), its hardware resources, and the set of instructions which execute on these resources. The hardware ressources are the set of the FUs $Q = \{q_1, \ldots, q_M\}$ such that N_q is the number of copies of the FU $q \in Q$. We associate to each instruction u its reservation table $\mathcal{RT}_u$.

3. DAG MODEL

A DAG $G = (V, E, \delta)$ consists of a set of operations V, and a set of arcs E which contains the data dependences between the operations with any other precedence constraints. Each operation u has a latency $\delta(u)$. We assume one sink operation $\perp$ in G which reflects the total schedule time: if there is more than one sink node, we add the virtual node $\perp$ with an arc e from each sink s to $\perp$ with $\delta(e) = lat(s)$. A valid schedule of G is a positive integer function σ which associates to each operation u an issue time $\sigma(u)$. Any acyclic schedule σ of G must ensure that:

$$\forall (u, v) \in E: \quad \sigma(v) - \sigma(u) \geq \delta(u)$$

In this paper, we consider that each operation $u \in V$ writes into at most one register of a type $t \in \mathcal{T}$. The operations which define multiple values with different types are accepted in our model iff they do not define more than one value of a certain type. We denote by u^t the value of type t defined by the operation u. We also consider the following sets:

1. $V_{R,t}$ is the set of the values of type $t \in \mathcal{T}$;
2. $E_{R,t}$ is the set of the flow dependency arcs through the values of type $t \in \mathcal{T}$. If there is some values not read in the DAG, or are still read after leaving this DAG, these values have to be kept in registers. We consider then that there is a flow arc from these values to $\perp$;

Finally, we consider that reading from and writing into a register can be delayed from the beginning of the schedule time (VLIW case). We define the two delay functions $\delta_{r,t}$ and $\delta_{w,t}$ such that:

$$\begin{array}{ll} \delta_{w,t}: & V_{R,t} \rightarrow \mathbb{N} \\ & u \mapsto \delta_{w,t}(u) / \ 0 \leq \delta_{w,t}(u) < \delta(u) \\ & \text{the write cycle of } u^t \text{ is } \sigma(u) + \delta_{w,t}(u) \\ \delta_{r,t}: & V \rightarrow \mathbb{N} \\ & u \mapsto \delta_{r,t}(u) / \ 0 \leq \delta_{r,t}(u) \leq \delta_{w,t}(u) < \delta(u) \\ & \text{the read cycle of } u \text{ is } \sigma(u) + \delta_{r,t}(u) \end{array}$$

4. ACYCLIC SCHEDULING PROBLEM

A valid schedule σ of G is first constrained by the inherent data dependency relations between the operations or any other serial constraints. The target architecture limitations impose other constraints which are the limited number of resources constraints and registers.

4.1 Resources Constraints

The resources constraints are simply the fact that two operations must not execute simultaneously on the same FU, i.e. the total number of operations which execute on a FU q during a clock cycle c must not exceed N_q the number of the FU copies. By using the reservation tables, an operation u executes on a FU q during a clock cycle c iff $\mathcal{RT}_u[c - \sigma(u), q] = 1$. Formally, the resources constraints are:

$$\forall 0 \leq c \leq \sigma(\perp), \ \forall q \in Q \qquad \sum_{u \in V} \mathcal{RT}_u[c - \sigma(u), q] \leq N_q$$

4.2 Registers Constraints

A value $u^t \in V_{R,t}$ is alive at the first step after the writing of u^t until its last reading (consumption). The set of the consumers of a value $u^t \in V_{R,t}$ is the set of the operations which read it:

$$Cons(u^t) = \{v / \ \exists (u, v) \in E_{R,t}\}$$

The last consumption of a value is called the killing date and noted;

$$\forall u^t \in V_{R,t} \quad kill(u^t) = \max_{v \in Cons(u^t)} \left(\sigma(v) + \delta_{r,t}(v)\right)$$

We assume that a value written at a clock cycle c in a register is available one step later. That is to say, if operation u reads from a register at a clock cycle c while operation v is writing in it at the same clock cycle, u does not get v's result but gets the value that was previously stored in that register. Then, the *lifetime interval* $LT^{\sigma}_{u^t}$ of the value u^t is $]\sigma(u) + \delta_{w,t}(u), kill(u^t)]$. Given the lifetime intervals of all the values, the number of registers of type t needed to store all the defined values is the maximum number of values of type t that are simultaneously alive. We call this number the register need (requirement) of the schedule σ, and we note it $RN^{\sigma}_t(G)$. This register need is computed by building the indirected interference graph $H^{\sigma}_t = (V_{R,t}, \mathcal{E})$, such that u^t and v^t are adjacent iff they are simultaneously alive, i.e. their lifetime intervals interfere. Then, the maximal number of values simultaneously alive is the cardinality of the maximal clique (complete subgraph) of H^{σ}_t.

Since the number $\mathcal{R}_t$ of available registers of type t is limited in the target processor, we need to find a schedule which doesn't need more than $\mathcal{R}_t$ registers:

$$\forall t \in \mathcal{T} \qquad RN^{\sigma}_t(G) \leq \mathcal{R}_t$$

If such schedule doesn't exist, spill code has to be generated, i.e. we must store some values in memory rather than in registers. Spilling increases the total schedule time because it inserts new operations and the spilled data may cause cache misses. We do not handle spill code in this paper.

4.3 Cache Effects

In the area of fine grain scheduling, the cache effects are rarely taken into account because their behavior differ from one platform to another. Furthermore, reducing the cache effects may require more registers to issue more operations during the miss stall cycles, and sometimes may require extensive code size expansion due to loop unrolling to exhibit more ILP. To exploit this ILP, the memory load which causes a cache misse must be issued well ahead of the operation which requires the loaded data in order to reduce the cache miss stall cycles to a minimum. The scheduling method used in this paper is based on this technique where we try to cover the compulsory misses if a subset of memory loads access to the same cache line.

Given some memory loads operations accessing the same cache line, the first issued load causes a cache compulsory miss and brings the entire line into the cache, while the subsequent access to the loaded cache line are hits. To fix the

ideas, we assume the following scenario. We call a leading cache effect[14] the penalty for a miss reference, and we note it lce[1]. A subsequent reference to the same cache line suffers a trailing cache effect tce due to the latency of fully servicing the miss : the requested data which causes the miss bypass the cache and goes directly from the memory bus to the CPU, while the subsequent hits must wait tce cycles for loading the whole cache line into the cache.

According to above, the cache effects make the memory operations latencies variable according to the schedule. There is an inter-dependence between the schedule and the cache effects. For instance, suppose that three memory loads a, b, c access the same cache line. If a is scheduled before b and c, then this load is an essential (compulsory) miss which can not be eliminated. The latency of a must be set to $\delta(a) = lce$ if we want to avoid stalling the processor. To eliminated the trailing cache effects of b and c, we must issue them after the schedule time of a with at least $(lce + tce)$ clock cycles.

5. INTEGER LINEAR PROGRAMMING

An integer linear programming problem (intLP) [11] is to solve :

$$\begin{cases} \text{maximize (or minimize) } cx \\ \text{subject to } Ax = b \end{cases}$$

with $c, x \in \mathbb{N}^n \ : \ x \geq 0$, and A is an $(m \times n)$ constraints matrix. This is the standard formulation. In fact, we can use other linear constraints $(\leq, \geq, <, >, =)$.

5.1 Logical Operators

Intrinsically, an intLP model defines the conjunctive operator $\wedge$. Given two constraints matrix A and A', saying that x must be a solution for both $Ax \geq b$ and $A'x \geq b'$ is modeled by :

$$\begin{pmatrix} A \\ A' \end{pmatrix} x \geq \begin{pmatrix} b \\ b' \end{pmatrix}$$

The nagation of a constraints matrix A with m lines (m linear constraints $f_1, f_2, \cdots, f_m$), i.e. forcing x to do not verify $Ax \geq b$ is modeled by :

$$f_1(x) < b_1 \ \vee \ f_2(x) < b_2 \ \vee \ \cdots \ \vee \ f_m(x) < b_m$$

In [6], the authors shown how to model the disjunctive operator $\vee$. Consider the problem :

$$\begin{cases} \text{maximize (or minimize) } f(x) \\ \text{subject to : } g(x) \geq 0 \ \vee \ h(x) \geq 0 \end{cases}$$

By introducing a binary variable $\alpha \in \{0, 1\}$, this disjunction is equivalent to :

$$\begin{cases} g(x) \geq \alpha \underline{g} \\ h(x) \geq (1 - \alpha)\underline{h} \end{cases}$$

where $\underline{g}$ and $\underline{h}$ are two known non null finite lower bounds for g and h resp. We generalize to an arbitrary number of constraints in an n-disjunctive formula $\vee_n$:

$$\vee_n(f_1, \cdots, f_n) = (f_1(x) \geq 0 \ \vee \ f_2(x) \geq 0 \vee \cdots \vee \ f_n(x) \geq 0)$$

Since the dichotomy operator $\vee$ is associative, we group the constraints two by two from left to right. There is $(n-1)$ internal $\vee$ operators which need to define $(n-1)$ boolean variables $(h_1, \cdots, h_{n-1})$. The final constraints system to express $\vee_n$ has $\mathcal{O}(n)$ constraints and $\mathcal{O}(n)$ boolean binary variables.

From above, we deduce the linear constraints of any other logical operator :

1. $g(x) \geq 0 \Longrightarrow h(x) \geq 0$ can be written
$$g(x) < 0 \vee h(x) \geq 0$$
2. $g(x) \geq 0 \Longleftrightarrow h(x) \geq 0$ can be written
$$\big(g(x) \geq 0 \wedge h(x) \geq 0\big) \vee \big(h(x) < 0 \wedge g(x) < 0\big)$$

5.2 Maximum an Minimum

In our intLP formulation, we need to compute the function $z = max(x, y)$ which can formulated by considering the following constraints :

$$\begin{cases} z \geq x \\ z \geq y \\ z \leq (1 - \alpha)x + \alpha\bar{y} \\ z \leq \alpha y + (1 - \alpha)\bar{x} \\ \alpha \in \{0, 1\} \end{cases}$$

where $(\bar{x}, \bar{y})$ are two finite non null upper bounds for x, y resp. We can also express the max_n function with arbitrary number of parameters $z = max_n(x_1, x_2, \cdots, x_n)$. Since max is associative, we group the variables two by two from left to right. The general form of max_n is :

$$\begin{cases} y_1 = max(x_1, x_2) \\ y_2 = max(y_1, x_3) \\ \vdots \\ y_{n-2} = max(y_{n-3}, x_{n-1}) \\ z = max(y_{n-2}, x_n) \end{cases}$$

We need to define $n-2$ intermediate y_i variables and $(n-1)$ systems to compute"max" operators. It leads to a complexity of $\mathcal{O}(n)$ intermediate and binary variables and $\mathcal{O}(n)$ linear constraints (each "max" operator needs 4 linear constraints to be defined).

Also, computing the minimum $z = min(x, y)$ can be done either by computing $z = -max(-x, -y)$ or by considering :

$$\begin{cases} z \leq x \\ z \leq y \\ z \geq (1 - \alpha)x + \alpha\underline{y} \\ z \geq \alpha y + (1 - \alpha)\underline{x} \\ \alpha \in \{0, 1\} \end{cases}$$

where $(\underline{x}, \underline{y})$ are two finite non null lower bounds for x, y resp. To express the min_n function, we also use the associativity of min by grouping the variables two by two from left to right as done for max_n. It leads to a complexity of $\mathcal{O}(n)$ intermediate variables (the binary variables and those which hold the intermediate minimums) and $\mathcal{O}(n)$ linear constraints.

If the domain sets of all the variables are bounded, the finite non null upper and lower bounds of all the linear functions are finite and can de determined [13].

6. EQUIMINMAX FORMULATION

In this section, we define a new formulation of scheduling problem using integer linear programming (intLP). We

[1]This latency depends on the memory access latency and the memory bus bandwidth.

named it EquiMinMax because it uses the linear constraints which express the equivalence relation ($\Longleftrightarrow$) and the functions min_n and max_n.

6.1 Basic Variables and Objective Function

For any operation $u \in V$, we define an integer variable σ_u which computes the schedule time. The objective function of our model is to minimize the total schedule time i.e.

$$Minimize\ \sigma_{\perp}$$

The first linear constraints describe the precedence relations. For any operation excluding the memory access, the latencies $\delta(u)$ are known statically. Let V_l be the set of memory (load) operations in G. The latency of these operations depends on the schedule time since this latter determines if a load is a compulsory miss or not. So, we need to define an integer variable δ_u for each load operation representing its latency which is set to a miss penalty lce iff u is a cache miss, and to a hit latency otherwise. The following precedence constraints are written in the model:

$$\begin{array}{ll} \forall e=(u,v) \in E \,/\, u \notin V_l & \sigma_v - \sigma_u \geq \delta(u) \\ \forall e=(u,v) \in E \,/\, u \in V_l & \sigma_v - \sigma_u \geq \delta_u \end{array}$$

There is $\mathcal{O}(|V|+|V_l|) \leq \mathcal{O}(2|V|)$ basic variables and $\mathcal{O}(|E|)$ linear constraints. To make the domains set of our variables bounded, we assume T as the worst possible schedule time by including in the model the constraint $\delta_{\perp} \leq T$. We chose T sufficiently large, where for instance the sum of all the operations latencies with considering a miss latency for loads is a suitable worst total schedule time[2]. As consequence, we deduce the bounded domain sets of our variables. For any $u \in V$:

- $\sigma_u \geq \underline{\sigma_u} = LonguestPathTo(u)$ is the "as soon as possible" schedule time;
- $\sigma_u \leq \overline{\sigma_u} = T - LonguestPathFrom(u)$ is the "as late as possible" schedule time according to the worst total schedule time T;

6.2 Registers Constraints

6.2.1 Interference Graph

The lifetime interval of a value u^t of type t is

$$LT_{u^t} =]\sigma_u + \delta_{w,t}(u),\ \max_{v \in Cons(u^t)} \big(\sigma_v + \delta_{r,t}(v)\big)]$$

We define for each value u^t the variable k_{u^t} which computes its killing date. The number of k_{u^t} variables is $\mathcal{O}(|V_{R,t}|)$. Since the domain of our variables is bounded, we know that k_{u^t} is bounded by the two following finite schedule times:

$$\forall t \in \mathcal{T}\ \forall u^t \in V_{R,t} \qquad \underline{k_{u^t}} < k_{u^t} \leq \overline{k_{u^t}}$$

where

- $\underline{k_{u^t}} = \underline{\sigma_u} + \delta_{w,t}(u)$ is the first possible definition date of u^t;
- $\overline{k_{u^t}} = \max_{v \in Cons(u^t)} \big(\overline{\sigma_v} + \delta_{r,t}(v)\big)$ is the latest possible killing date of u^t.

[2]The case where no ILP is exploited.

We use the max_n linear constraints to compute k_{u^t} like explained in Sect. 5.2: we need to define for each k_{u^t} $\mathcal{O}(|Cons(u^t)|)$ variables and $\mathcal{O}(|Cons(u^t)|)$ linear constraints to compute it. The total complexity to define all the killing dates is bounded by $\mathcal{O}(|V|^2)$ variables and $\mathcal{O}(|V|^2)$ constraints.

Now, we can consider H_t the indirected interference graph of G for the register type t. For any couple of values of the same type $u^t, v^t \in V_{R,t}$, we define a binary variable $s^t_{u,v} \in \{0,1\}$ which is set to 1 if the two values lifetimes intervals interfere: $\forall t \in \mathcal{T},\ \forall$ couple $u^t, v^t \in V_{R,t}$

$$s^t_{u,v} = \begin{cases} 1 & \text{if } LT_{u^t} \cap LT_{v^t} \neq \phi \\ 0 & \text{otherwise} \end{cases}$$

For any registers type $t \in \mathcal{T}$, the number of variables $s^t_{u,v}$ is the number of combinations of 2 values among $|V_{R,t}|$ i.e. $\big(|V_{R,t}| \times (|V_{R,t}|-1)\big)/2$.

$LT_{u^t} \cap LT_{v^t} = \phi$ means that one of the two lifetime intervals is "before" the other, i.e. $LT_{u^t} \prec LT_{v^t} \vee LT_{v^t} \prec LT_{u^t}$ where $\prec$ denotes is the precedence operator ("before") in the interval algebra. Then, we have to express:

$$s^t_{u,v} = 1 \Longleftrightarrow \neg\big(LT_{u^t} \prec LT_{v^t} \vee LT_{v^t} \prec LT_{u^t}\big)$$

Since $s^t_{u,v} \in \{0,1\}$, these constraints are equivalent to:

$$s^t_{u,v} \geq 1 \Longleftrightarrow \begin{cases} k_{u^t} - \sigma_v - \delta_{w,t}(v) - 1 \geq 0 \\ k_{v^t} - \sigma_u - \delta_{w,t}(u) - 1 \geq 0 \end{cases}$$

Given three logical expressions (P, Q, S), $(P \Longleftrightarrow (Q \wedge S))$ is equivalent to $(P \wedge Q \wedge S) \vee (\neg P \wedge \neg Q) \vee (\neg P \wedge \neg S)$. We write these two disjunctions with linear constraints by introducing two binary variables $h, h' \in \{0,1\}$ (see Sect. 5.1) and computing the finite non null lower bounds of the linear functions. The complexity of computing all the $s^t_{u,v}$ variables is $\mathcal{O}\big(|V_{R,t}| \times (|V_{R,t}|-1)\big)$ binary variables (two for each couple of values (u^t, v^t)) and $\mathcal{O}\big(7/2|V_{R,t}| \times (|V_{R,t}|-1)|\big)$ linear constraints (seven for each couple of values). The total complexity of considering the interference graph H_t is then bounded by $\mathcal{O}(|V_{R,t}|^2)$ variables and $\mathcal{O}(|V_{R,t}|^2)$ constraints.

6.2.2 Maximal Clique in the Interference Graph

The maximum number of values of type t simultaneously alive corresponds to a maximal clique in $H_t = (V_{R,t}, \mathcal{E}_t)$, where $(u^t, v^t) \in \mathcal{E}_t$ iff their lifetime intervals interfere ($s^t_{u,v} = 1$). For simplicity, rather to handle the interference graph itself, we prefer considering its complementary graph $H'_t = (V_{R,t}, \mathcal{E}'_t)$ where $(u^t, v^t) \in \mathcal{E}'_t$ iff their lifetime intervals do *not* interfere ($s^t_{u,v} = 0$). Then, a maximal clique in H_t corresponds to a maximal independent set[3] in H'_t.

To write the constraints which describe the independent sets (IS), we define a binary variable $x_{u^t} \in \{0,1\}$ for each value $x_{u^t} \in V_{R,t}$ such that $x_{u^t} = 1$ iff u^t belongs to an IS of H'_t. We must express in the model the following linear constraints: $\forall t \in \mathcal{T},\ \forall$ couple $x_{u^t}, x_{v^t} \in V_{R,t}$

$$x_{u^t} + x_{v^t} \leq 1 \Longleftrightarrow s^t_{u,v} = 0$$

We use the linear expressions of the equivalence ($\Longleftrightarrow$) defined in Sect. 5.1 by introducing a variable $h \in \{0,1\}$. The number of the variables x_{u^t} is $\mathcal{O}(|V_{R,t}|)$. The number of introduced binary variables to express the equivalences is $\mathcal{O}(1/2 \times |V_{R,t}| \times (|V_{R,t}|-1))$. The number of the linear constraints to define the IS is $\mathcal{O}(2 \times |V_{R,t}| \times (|V_{R,t}|-1))$.

[3]It is a subgraph such that there is no two adjacent nodes.

The registers constraints are the fact that any set of values simultaneously alive must not exceed the number of available registers $\mathcal{R}_t$. Thereby, we write in the model:

$$\forall t \in \mathcal{T} \qquad \sum_{u^t \in V_{R,t}} x_{u^t} \leq \mathcal{R}_t$$

There is $\mathcal{O}(|\mathcal{T}|) = \mathcal{O}(1)$ such constraints.

6.3 Cache Effects

In this section, we show how to model the compulsory cache misses and how they influence the schedule. We start by grouping the memory access operations into subsets $V_{l_i} \subseteq V_l$, such that all the operations belonging to the same subset V_{l_i} access to the same cache line i (according to the cache line boundaries [10]). so $V_{l_0} = \{a, b, c\}$. The first issued load in a subset V_{l_i} causes a cache miss. Its latency must be changed to lce. The remaining operations within that subset have a hit latency while their issue time must be delayed at least with $(lce + tce)$ like explained in Sect. 4.3.

To identify which load operation is being scheduled first and causes a miss, we define a variable m_i for each subset V_{l_i} which holds the first (minimal) issue time:

$$\forall V_{l_i} \subseteq V_l \qquad m_i = \min_{u \in V_{l_i}} \sigma_u$$

We use the linear expression of min_n explained in Sect. 5.2. There is at most $\mathcal{O}(|V_l|)$ m_i variables. The number of constraints and variables to compute all the m_i is bounded by $\mathcal{O}(|V_l|) \leq \mathcal{O}(|V|)$.

Any memory access operation $u \in V_{l_i}$ scheduled at time m_i must have a miss latency to avoid stalling the processor. We write in the model the linear constraints of:

$$\forall V_{l_i} \subseteq V_l,\ \forall u \in V_{l_i} \qquad (\sigma_u = m_i) \Longrightarrow (\delta_u = lce)$$

All the subsequent memory access operations in V_{l_i} are hits and must be delayed to avoid the trailing edge effects. We write in the model the linear constraints of: $\forall V_{l_i} \subseteq V_l$

$$\forall u \in V_{l_i} \qquad (\sigma_u > m_i) \Longrightarrow \begin{cases} \delta_u = hit \\ \sigma_u - m_i \geq lce + tce \end{cases}$$

The number of the linear constraints which describe all these implications is bounded by $\mathcal{O}(|V_l|) \leq \mathcal{O}(|V|)$.

6.4 Resources Constraints

6.4.1 Conflicting Graph

The resources constraints are handled by considering for each FU an indirected graph $F_q = (V, \mathcal{E}_q)$ which represents the conflicts between the instructions on a FU $q \in Q$. For any couple of operations, $(u, v) \in \mathcal{E}_q$ iff u and v are in conflicts on q. Any clique in F_q represents the set of operations which conflict on q at the same time. So, any clique must not exceed N_q the number of copies of the FU q.

We define a binary variable $f^q_{u,v} \in \{0, 1\}$ such that $f^q_{u,v} = 1$ iff there is a conflict between u and v on the FU q. Given the RT of two operations u and v, we can deduce when a structural hazards occurs on the FU q. The general formulation of the conflicting variables is the disjunction of all the cases where a conflict on the FU occurs.

Let $U_{u,q}$ be the set of clock cycles in the reservation table of u where the FU q is used by u:

$$\forall u \in V\ \forall q \in Q \qquad U_{u,q} = \{c \in \mathbb{N} / \mathcal{RT}_u[c, q] = 1\}$$

The set of all cases where two operations conflicts on a FU q are described by the cartesian product $U_{u,q} \bigotimes U_{v,q}$. The general formula of the binary conflicting variables is then: $\forall q \in Q\ \forall$ couple $u, v \in V$

$$f^q_{u,v} = 1 \Longleftrightarrow \bigvee_{(c1,c2) \in (U_{u,q} \bigotimes U_{v,q})} \sigma_u + c1 = \sigma_v + c2$$

We use the linear constraints of equivalences and disjunctions defined in Sect. 5 to write the linear description of this formula in the model. The number of terms in this disjunction depends on $|U_{u,q} \bigotimes U_{v,q}|$ which is a function of the target architecture characteristics (reservation tables and instructions set), and thereby it is constant for the input DAGs.

6.4.2 Maximal Click in the Conflicting Graph

For simplicity, rather than considering the conflict graph F_q itself, we use its complementary $F'_q = (V, \mathcal{E}'_q)$ such that $(u, v) \in \mathcal{E}'_q$ iff u and v are *not* in conflicts on q ($f^q_{u,v} = 0$). Then, a clique in F_q becomes an independent set in F'_q.

We define a binary variable $y^q_u \in \{0, 1\}$ for each operation u such that $y^q_u = 1$ iff u belongs to an IS of F'_q. We write in the intLP model the linear constraints of IS:

$$\forall q \in Q\ \forall \text{ couple } u, v \in V \qquad y^q_u + y^q_v \leq 1 \Longleftrightarrow f^q_{u,v} = 0$$

We use the linear constraints of the equivalence (Sect. 5.1) by introducing a binary variable $h \in \{0, 1\}$. There is $\mathcal{O}(1/2 \times |V| \times (|V| - 1))$ binary variables h for each FU (one for each couple of operations) and $\mathcal{O}(2 \times |V| \times (|V| - 1))$ linear constraints to describe the IS. The resources constraints are the fact that the cardinality of the any independent set in F'_q must not exceed N_q. We write in the model:

$$\forall q \in Q \qquad \sum_{u \in V} y^q_u \leq N_q$$

There is $\mathcal{O}(|Q|) = \mathcal{O}(1)$ such linear constraints.

7. RELATED WORK AND DISCUSSION

Acyclic scheduling under registers and resources constraints is a classical problem where lot of works have been done. An intLP formulation (SILP) was defined in [15] to compute an optimal schedule with register allocation under resources constraints. The complexity of this model is bounded by $\mathcal{O}(|V|^2)$ variables and $\mathcal{O}(|V|^2)$ constraints. However, this formulation does not introduce registers constraints, i.e. it does not limit the number of values simultaneously alive. Other formulations [7, 9] introduced registers constraints. The number of variables was $\mathcal{O}(|V|^2)$ but the number of the linear constraints grown exponentially due to registers constraints.

A polynomial formulation for the registers constraints was defined in [4] with a complexity of $\mathcal{O}(T \times |V|)$ variables and $\mathcal{O}(|E| + T \times |V|)$ constraints. Similar approaches minimized the register requirement for the exact cyclic scheduling problem (software pipelining) under registers and resources constraints [1, 5, 3]. It is easy to rewrite these intLP models to solve the acyclic scheduling problem. All these formulations had a complexity which depended on the worst total schedule time T. Indeed, they define a binary variable $\sigma_{u,c}$ for each operation u and for each execution step c during the whole execution interval $[0, T]$. $\sigma_{u,c}$ is set to 1 iff the operation u is scheduled at the clock cycle c. The complexity of

their models was clearly bounded by $\mathcal{O}(T \times |V|)$ variables and $\mathcal{O}(|E| + T \times |V|)$ constraints. In fact, the factor T can be very large in real codes since it depends on the input data itself (critical paths and specified operations latencies). We think that a complexity must depend only on the *amount* of input data and not on the date itself. Otherwise, the resolution time would not scale very well. For instance, if a memory operation is always a cache miss, then we change its static specified latency to a memory access (~ 100) in order to better exploit free slots during scheduling. The number of variables and constraints generated with all these techniques is multiplied by a factor of hundred, while the size of our model does not change anymore.

The coefficients introduced by our formulation in the final constraints matrix are all bounded by T and $-T$, which is the case of the coefficients in the models defined in [1, 4, 5, 4]. If T is very huge, the resolution process can be difficult because of computational overflows [11]. Since EquiMinMax reduces the size of the model, resolving an EquiMax model is less critical than any one of the cited techniques.

8. CONCLUSION

In this work, we give an intLP formulation of the optimal scheduling under resources and registers constraints with cache effects. The FUs can have a complex and heterogeneous usage pattern and are modeled by reservation tables. We handle multiple registers types and delayed read from and write into the registers. In this work, we reduce the cache effects caused by the compulsory misses. The complexity of our model is polynomial on only the size of the input DAG. Theoretically, our formulation must reduce considerably the exact resolution time. In the future, we will try to model the capacity and conflict misses and extend our formulation to cyclic scheduling (software pipelining), where the lifetime intervals of the values and the resources usage patterns become cyclic.

9. REFERENCES

[1] Eric Altman. *Optimal Software Pipelining with Functional Units and Registers.* PhD thesis, McGill University, Montreal, October 1995.

[2] Alain Darte, Yves Robert, and Frédéric Vivien. *Scheduling and Automatic Parallelization.* Birkhauser Boston , 2000.

[3] Alexandre E. Eichenberger, Edward S. Davidson, and Santosh G. Abraham. Minimizing Register Requirements of a Modulo Schedule via Optimum Stage Scheduling. *International Journal of Parallel Programming*, 24(2):103–132, April 1996.

[4] Christine Eisenbeis, Franco Gasperoni, and Uwe Schwiegelshohn. Allocating Registers in Multiple Instruction-Issuing Processors. In *Proceedings of the IFIP WG 10.3 Working Conference on Parallel Architectures and Compilation Techniques, PACT'95*, pages 290–293. ACM Press, June 27–29, 1995.

[5] Christine Eisenbeis and Antoine Sawaya. Optimal Loop Parallelization under Register Constraints. In *Sixth Workshop on Compilers for Parallel Computers CPC'96.* , pages 245–259, Aachen - Germany, December 1996.

[6] Robert S. Garfinkel and George L. Nemhauser. *Integer Programming.* John Wiley & Sons, New York, 1972. Series in Decision and Control.

[7] C. H. Gebotys. Optimal Scheduling and Allocation of Embedded VLSI Chips. In *Proceedings of the 29th Conference on Design Automation*, pages 116–119, Los Alamitos, CA, USA, June 1992. IEEE Computer Society Press.

[8] C. H. Gebotys and M. I. Elmasry. A Global Optimization Approach for Architectural Synthesis. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 258–261, Santa Clara, CA, November 1990. IEEE Computer Society Press.

[9] D. Kaestner and M. Langenbach. Code Optimization by Integer Linear Programming. *Lecture Notes in Computer Science*, 1575:122–136, 1999.

[10] David A. Patterson and John L. Hennessy. *Computer Organization and Design The Hardware-Software Interface.* Morgan Kaufmann Publishers, 1994.

[11] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1987.

[12] Jurij Silc, Borut Bobic, and Theo Ungerer. *Processor Architecture: from Dataflow to Superscalar and Beyond.* Springer, first edition, 1999.

[13] Sid-Ahmed-Ali Touati. Optimal Register Saturation in Acyclic Superscalar and VLIW Codes. Research Report, INRIA, November 2000. ftp.inria.fr/INRIA/Projects/a3/touati/optiRS.ps.gz.

[14] Perry Wang and Edouard Davidson. Hierarchical performance modeling with cache effects: A case study of the dec alpha. Technical report, Advanced Computer Architecture Laboratory, University of Michigan, March 1995. http://www.eecs.umich.edu/home/techreports/cse95.html.

[15] L. Zhang. *SILP: Scheduling and Register Allocation with Integer Linear Programming.* PhD thesis, University of Saarlands, 1996.